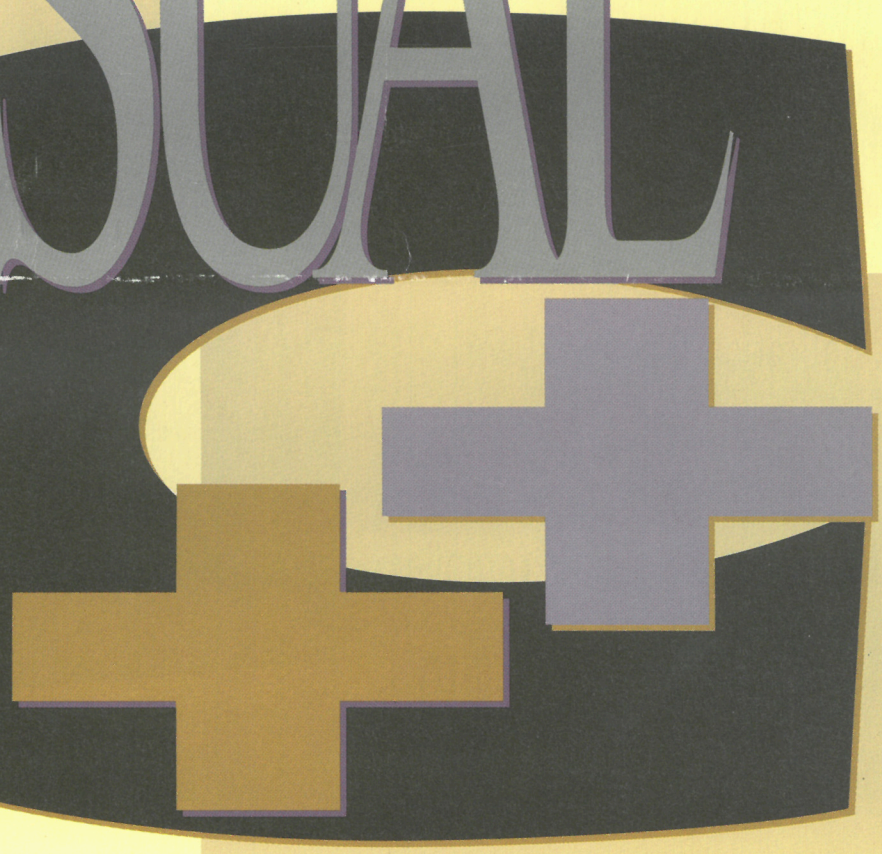
Presenting

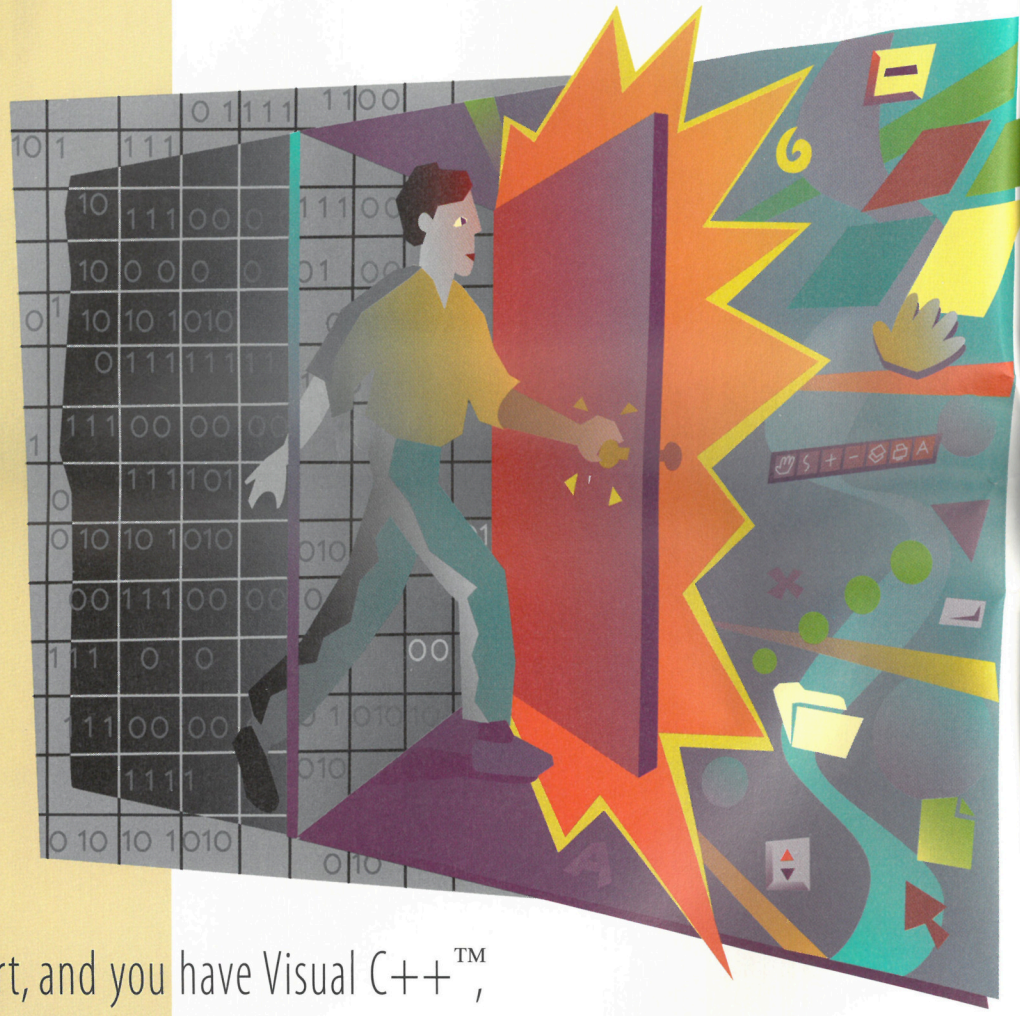
Microsoft®

VISUAL



//in our own words

Visually Powerful



WHAT'S NEW?

Better integration, new WYSIWYG tools, and a mature application framework for Windows™. That's what's new. Mix with world-class compiler technology, plus the industry's best documentation and support, and you have Visual C++™, the premier application development system for Windows.

The smartest thing we did was to listen to our customers. The rest was pretty straightforward.

AUDREY WATSON

"From a product standpoint, Visual C++ almost designed itself, because when we talked to our customers, the common needs really jumped out. "I need to move to C++, but I don't have time. My customers want Windows apps, but Windows programming is too complex. So, make your product easier to use. Help me deliver applications faster. Speed up your tools but don't take away power. And give me a lot of flexibility."

That's a big shopping list, but the amazing thing is, we had the right people and technology to answer these needs in Visual C++.

The Visual Workbench gives you vastly better integration. Your tools are all right there, and they work together naturally. So by integrating the toolset, we streamlined the whole process of programming for Windows.

And, we added **important new tools.** App Studio gives you WYSIWYG resource editing and it can use VB custom controls. The AppWizard quickly generates a working skeleton application. And ClassWizard lets you easily connect code to your user-interface objects.

So, better tools save time, but the coding issue remains. How can we reduce the sheer complexity of writing Windows programs? That's where the **Microsoft Foundation Class libraries** come in. You deliver applications faster because you're not starting from scratch, with the Windows API. You start with a robust, application framework that has all kinds of reusable components.

Finally, underlying it all are **world-class support components.** There's innovative online help and comprehensive print documentation. There's sample code, Microsoft Developer Network, and companion products like Microsoft Test. There's customer support, and more."

TOC

TABLE OF CONTENTS

Visual C++

Run SETUP.EXE

WHAT ELSE CAN WE SAY?

Setup for Visual C++ is faster and easier than ever before. We speeded up its execution time, and made the user interface as simple and intuitive as possible, so you can get up and running right away.

For more information on installation, see Chapter 2 of the *Visual Workbench User's Guide*.

"We worked really hard to streamline Setup for Visual C++. It's a single application, hosted in Windows, very easy to use. If you want the default installation, you just run Setup, hit Continue three times in a row, and away you go. Don't forget to type in your registration information, of course, and you'll want to restart Windows in most cases, to enable the new settings.

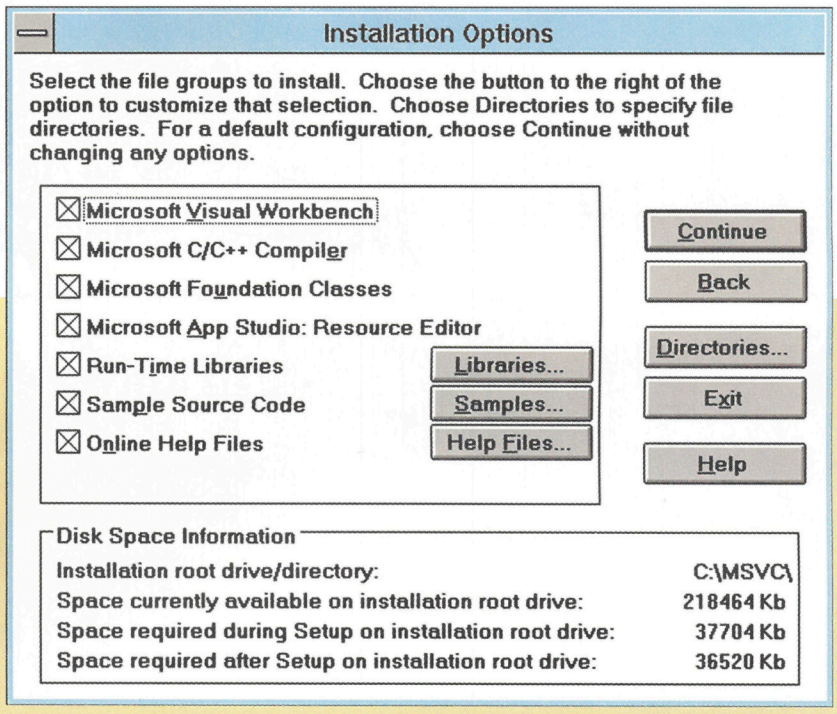
The default installation lets you try out Visual C++ immediately. You can test-drive the new features like App Studio, build some sample apps right out of the box, and so on. Some customers had trouble building samples in previous releases. **The default setup ensures you have a working installation right off the bat.**

Or, if you need a custom setup, you can easily change whatever part of the installation you want. We tried to make Setup basically self-documenting. Choose from what you see, click OK, and forget about it. Most people probably want Setup to work like a bank card machine — it's like, "Keep out of the way and just give me the money."

And remember, you can always run Setup again. Maybe you change your mind about something, or you need a different setup for a new project, that's no problem. Just pop in the disk and fire it up.

We invested development time in speeding up Setup because customers are impatient during installation. Our Setup looks and acts like a regular Windows setup program, but we wrote custom code to eliminate delays while Setup builds file lists. And now we're very accurate in judging the amount of required disk space. We check the cluster size on the disk you're installing to and adjust the file sizes we're installing accordingly. So Setup is a lot more precise in estimating what kind of resources we're going to take on your system — very important because of the size of this product."

Our new Setup program makes installation easier than ever.



Setup is fun to work on. It's one thing that nobody can escape, so it certainly gets a lot of feedback. And it's the very first product component that the customer sees. First impressions do matter. And that means Setup has to be good. We think it is.

PAT BRENNER

Setup

Cutting Edge

THE INTEGRATED DEVELOPMENT ENVIRONMENT

Visual Workbench contains all the tools necessary for you to develop a program. From this central point, you can undertake all the tasks required: you can edit source code, create visual objects, connect these objects to code, build your application, and debug the program.

For more information on Visual Workbench, see the *Visual Workbench User's Guide*.

"I think the most intriguing aspect of Visual Workbench is that it's integrated. You have an editor, a debugger, a browser, online Help, all available within the same environment, in the same place, so you don't have to go out to an MS-DOS® box or go to CodeView® for Windows or go to the documentation that's off on your desk somewhere. You can use the entire product on a portable. You can be outside, writing an application, and you don't need to have a bunch of other stuff around—you can do it all within the one environment.

The other really compelling thing about the Visual Workbench is that it is concurrent. For instance, you can build and edit at the same time. You can use the browser while you debug your application. **You can do a multiplicity of things simultaneously within the same environment.** This is going to make you more productive, because if, for example, you're debugging and you need to change some code, you don't have to drop out of your debugger; you can just change the code right there.

Where the Visual Workbench really shines is when it takes your source files from other projects and actually turns them into an executable very rapidly. The steps are very simple.

First, you open up a new project, and we have a series of templates that allow you to set the correct compiler and linker defaults for a variety of applications—Windows applications, applications with QuickWin libraries, DLLs, VB custom controls, and so on.

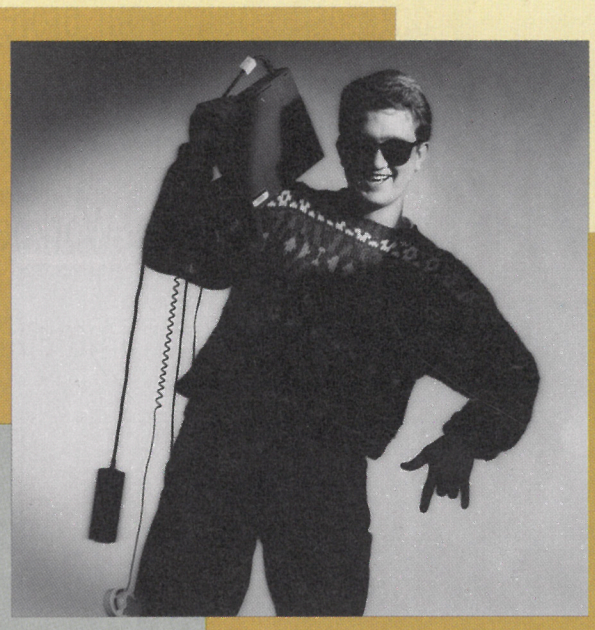
Next, you add all of your source files to your project. It's really straightforward in our Edit Project dialog box to select all the files you want to add and then add them directly into your project.

You're then free to either build your application with the defaults we provide or you can actually go in and modify or customize those defaults. **And we give you full control over what options are set for the source files**—for example, you can set precompiled headers, the memory model, the code generation…you name it!

After you've done that, **building your program is just a push-click away.** On the toolbar we've got an icon which allows you to rebuild your entire project. Just click on that, and the build will take place in the background.

After the build is successfully completed, you can browse your application and take a look at the symbols that are in the application. You can also start debugging the application. You can set a variety of types of breakpoints, you can set location breakpoints or breakpoints on particular variables—sometimes they call those watchpoints. And we've got a button on the toolbar which allows to you to just start your application running."

What we concentrated on the most with this development environment was to make the common tasks simple and the complex tasks possible, so you work without the environment getting in your way. We've automated a lot of the tasks that have previously caused developers some heartache and lots of work, and things like creating projects or setting options or browsing or debugging are all much simpler than they've ever been in the past. We're really excited about our new Windows-hosted development environment. It's something that we think developers are going to love and use for many years to come.



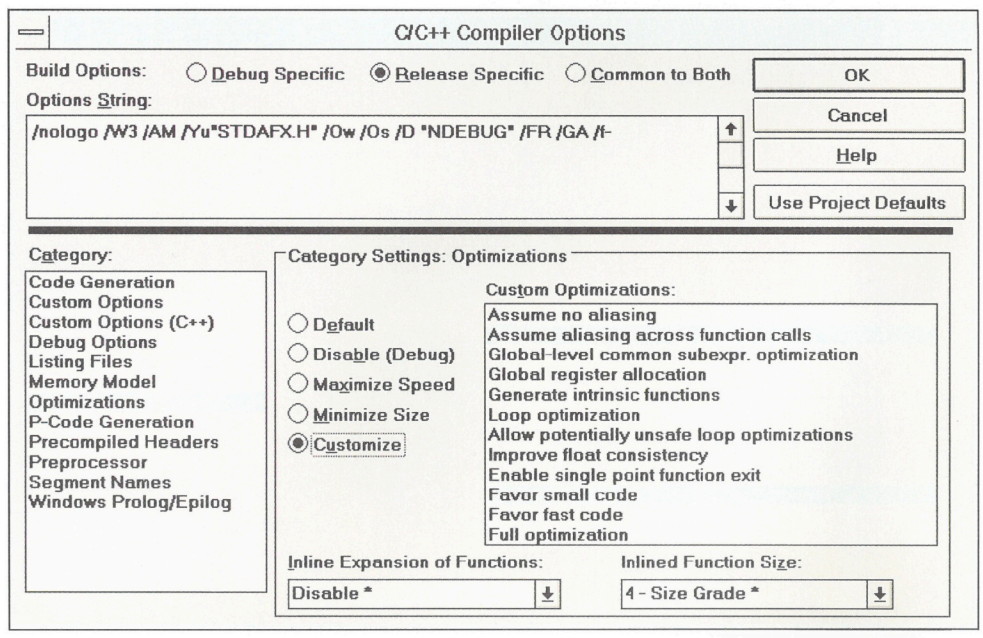
JEFF BEEHLER

Visual Workbench

Graphical

POINT-AND-SHOOT PROJECTS

Visual Workbench projects help you organize your program development. Project types provide the correct default compiler and linker options to build your project. Include files are automatically scanned and added to the project dependency list. Workspaces let you save and recall your own personalized Visual Workbench layouts or automatically recall the last layout you used. And you can immediately build your current projects without modification as Visual Workbench external projects.



The Compiler Options dialog makes it easy to browse various options and choose just what you need.

For more information on projects, see Chapter 8 of the *Visual Workbench User's Guide*.

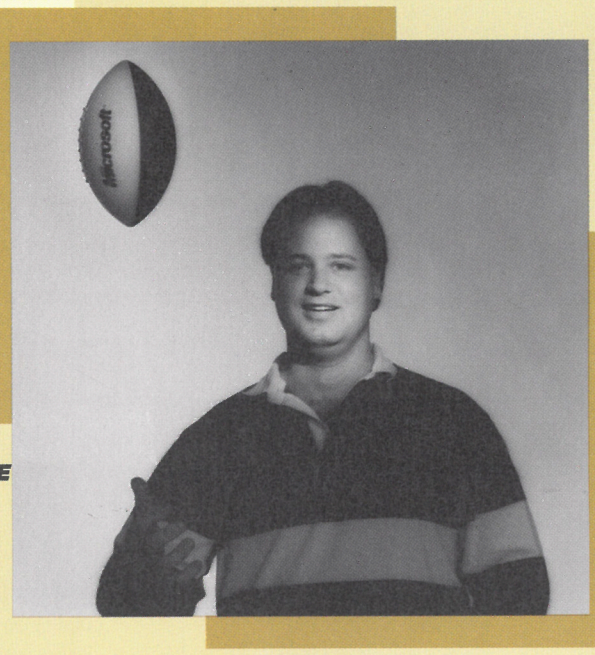
"One of the nice things about Visual Workbench is that it allows you to specify what kind of target you're going to build. It can be a Visual Basic custom control, a Windows .EXE, a Windows DLL, a DOS application, whatever ... it'll generate the makefile necessary for that, including the correct switches and correct libraries, in order to build whatever target you want. Also, in generating the makefiles, it will scan for your include dependencies and your RC file dependencies, so if you were to change one of those before you build, it'll recognize that those have changed and will build only those that use those files for you. It makes it a lot easier to create the makefiles than to create them by hand.

Sometimes you might need to change the compiler and linker options that Visual Workbench gives you — for example, your particular project requires options that we don't provide as a default setting for whatever your target is. **Our dialog boxes for the compiler options and linker options have a new changing pane for easy access to specific catagories and switches.** For instance, if you have code-generation-type switches, part of the dialog box will actually change for you and present you all the code-generation switches. Optimization — it will change for that also. And in the linker options, one such category contains the output specifications, like if you want more information in your linker output or map files or whatnot, it can file those things for you within these dialog boxes. There are also several Windows import libraries that we provide that are not included by default, such as some of the common dialog-type links. We give you a large list of those, and just by clicking on it, we'll include those libraries on the option line.

If you have makefiles from projects before Visual C++ that you don't want to change, we still support those, and you can build those projects within Visual Workbench, but you also have the ability to specify your debug target. For instance, if you're going to debug a Windows .EXE, you can take advantage of the integrated debugger within the Visual Workbench with your external project. So you're still able to use a large percentage of the environment, even though your makefile is not generated by Visual Workbench.

Another feature that helps you organize projects is workspaces, which are a way to save your window layout for specific tasks. For instance, the ones that we provide for — that is, the ones that we have access to on the menu — are, by default, Edit, Debug and Custom. And the way I personally use those is that I have a window-layout form in my edit session, where I'll have my edit source in the top two-thirds of the screen, and in the bottom third I'll have my Browse window. And then I can save that and whenever I want that layout, with just a simple keystroke I can actually bring up that window layout automatically. I also have one for my debug session, with the Watch window and the Registers window and the other debug windows arranged the way I like 'em. So when I do a debug, with just another keystroke I can bring that up.

The third one I have is for when I'm building, the Custom one. It's just like my edit layout except, instead of the Browse window at the bottom, I have my Output window so I can keep tabs on what's happening with my build while I'm working on my project. People around here also use the Last Workspace Used item, so when you close your project or you close Visual Workbench, it will save the window layout as it was when you left, so next time you open up that project, or next time you bring up the Workbench, it will restore those windows in that same configuraion."



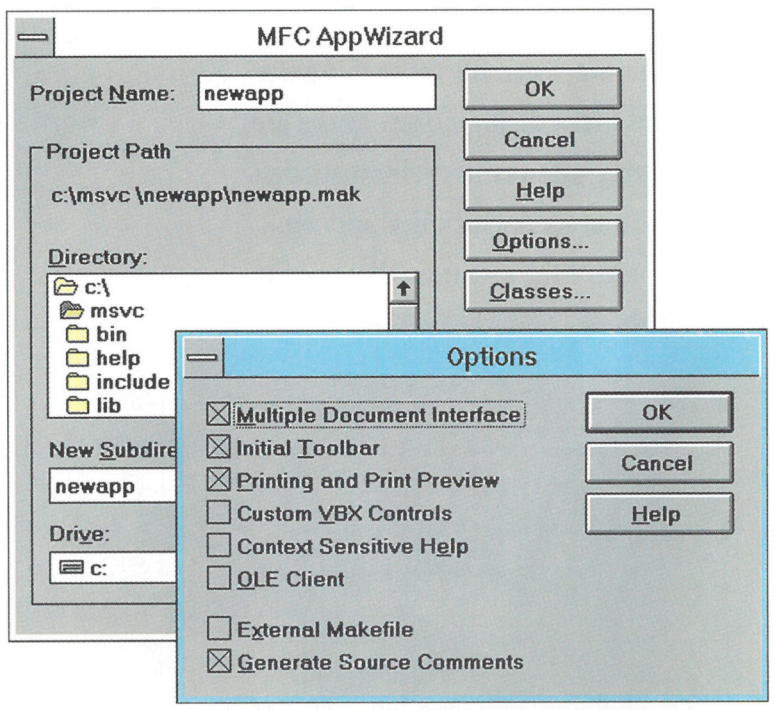
TOM WHITE

Projects

AppWizard, ClassWizard

PROGRAMMER'S APPRENTICE

AppWizard and ClassWizard are Windows programming assistants that orchestrate common programming tasks, such as connecting code to user-interface elements or starting a new Windows programming project. Wizards reduce the time you spend in repetitive coding, and free you to concentrate on what your application really needs to do.



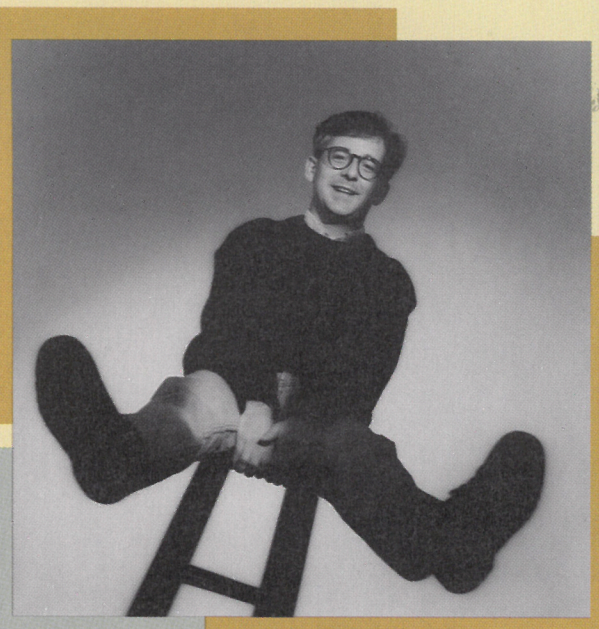
AppWizard lets you start with a working program that includes professional features such as a toolbar and print preview.

For more information on Wizards, see Chapter 9 of the *App Studio User's Guide* and Chapters 2, 6, and 7 of the *Class Library User's Guide*.

"We have two Wizards in Visual C++: AppWizard and ClassWizard. A Wizard is like a programmer's assistant that helps you by automating some of the more mundane tasks of programming. Both of our Wizards are designed to work with the Microsoft Foundation Class library, and you can access them from Visual Workbench and App Studio.

AppWizard helps when you're starting a new Windows application. The old way to start out is to copy a bunch of files from GENERIC, or some other simple Windows application, and then rename and modify them for your new project. **AppWizard automates that first step,** and beyond that, it lets you include a whole host of features in your application. You just pick from a list to include major features in your application, like a toolbar and status bar, printing and print preview, VBX support, OLE, context-sensitive help, and so on. Within minutes, you're ready to build, and you can start with a lot more than "Hello World" for Windows. You have the basis of a professional Windows application, with major features, and it's all tight, error-free code you won't have to worry about. The application conforms to the standard Windows user-interface model as well. But most of all, the application leverages MFC. **Each class in an AppWizard application is derived from one of the reusable MFC classes.** So, for example, if your application uses MDI, then AppWizard gets you started by deriving from the MFC class CMDIFrameWnd.

ClassWizard is for later in the coding process when you've designed your interface, your application design is in place, and you know what you want your application to do. It's a lot of work to connect up these things. Like, you just bought a big electronics kit, and you're an experienced engineer. You know where everything should go, but it's still **a lot of hand work to connect it all together.** That's what ClassWizard is for. In a typical MFC Windows app, you have generally three types of things to connect. You have user-interface elements, you have Windows messages, and you have your C++ application code. You know exactly what connections you need to make, but again, doing it all by hand is tedious and error-prone, since it's basically hand-editing a bunch of source files. **With ClassWizard, you connect these elements just by pointing and clicking.** So if you want a particular dialog control to handle a particular message, just point and click and ClassWizard lets you jump straight to the function definition and you can start entering your application-specific code. ClassWizard also integrates with the MFC Dialog Data Validation and Exchange mechanism, so it's easy to implement custom forms. It's a graphical process, like so many other things you do in Visual C++, and it's much quicker and more reliable."



STEVEN SINOFSKY

"Wizards are the kind of thing you use once, and say, "Wow! I can't believe nobody did this years ago."

The basic idea is very simple — just automate common programming tasks, and there are a lot of those in Windows. The Wizards free you from drudgery so you can spend your time on the things that really make your application different. We like to call them "programming assistants" because they do exactly what you'd do yourself, only faster and without errors. But you always have the flexibility to go in and edit the code yourself. The Wizards work the way you do, using standard C++, Windows, and MFC idioms, so there are no special tools or hard-to-understand conventions to follow. So they're great for the experienced Windows developer as well as someone who's just starting out.

Wizards

Foundation

STATE-OF-THE-ART FRAMEWORK FOR WINDOWS

The Microsoft Foundation Class Library (MFC) is a library of C++ classes that gives you the framework of an application for Windows. You get solid implementations of common features such as dialogs, splitter windows, and toolbars. You can easily build upon the framework, while overriding parts of it that you want to customize.

For more information on the Microsoft Foundation Class Libraries, see the *Class Library User's Guide* and the *Class Library Reference.*

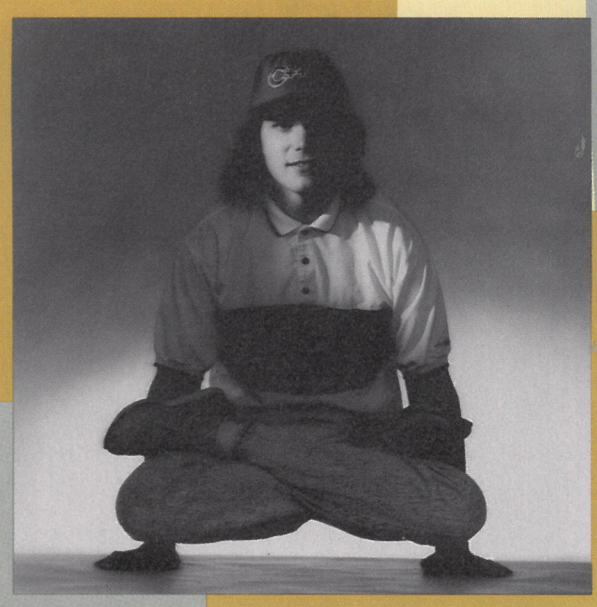
"What is the Microsoft Foundation Class Library? Well, it's really just a framework for building Windows applications, making it easier to build them using C++. It's an architecture that guides you down the right paths and gives you a lot of ways of doing common things, so it's easy to build the infrastructure of the application. But it's also scalable, so you can add additional things on top of it. We give you some basic parts you can build with, and you can do your own thing with them and on top of them.

Also, the Microsoft Foundation Class Library is C++. **We are an object-oriented class library and application framework.** If you're familiar with Windows terminology, you can actually get a good feeling for what it is by looking at the class hierarchy diagram, even though you haven't read all our documentation or memorized our class library reference. It's not meta-levels of abstraction ... if you look at a class when you read the name, you can pretty much figure out what it does.

Finally, the Microsoft Foundation Class Library is the C++ API to Windows. That's really the heart of our product, and **what we're trying to do is be the C++ interface to the underlying Windows operating system, while at the same time making it easier to use** that underlying Windows operating system. That's why Visual C++ is different from C7. It's now a visual-centric product. The target platform here is Windows. You can technically build DOS programs, but that's not the focus of the product because more and more people are building Windows apps, and you're using Windows-hosted tools to build a Windows app.

After doing MFC version 1, we didn't throw it out and start over for version 2. We'd already done our prototypes and thrown them away during the R & D phase before we started MFC version 1. So version 2 was built on top of version 1 ... we found that many of the version 1 classes worked really well for doing what we wanted to add in version 2. Some of the more complicated components of version 1 got simpler and components that didn't exist in version 1 are now there and were easy to add *because* of the infrastructure provided by version 1. If you've got an MFC version 1 app, you won't need to change hardly any of your code. Just as MFC doesn't require changing hardly any of your C Windows code.

Really, it comes down to this: MFC is the hub of what you're trying to do. Once your development is done, you leave your copy of the Visual Workbench and App Studio and all the Wizards on your development machine. Your customer gets a part of the class library with each copy of your application. And it just so happens that our tools are built with our class library. For example, App Studio is written in C++ using MFC version 2. It was built using the Visual Workbench and App Studio, of course — so we use our tools to build our tools."



What makes MFC unique versus all the other class libraries out there, I think, is that we are a serious application framework. We don't compromise efficiency. You don't have to read someone's thesis to understand our framework. The worst criticism we have heard about MFC version1 is that we are the obvious thing to do on top of Windows. People think that's a criticism because they expect application frameworks to be grandiose, abstract things that hide all the details from you. Actually, to avoid a lowest common denominator approach, we exploit the fact that you can get to the details when you want to, but you don't have to see them if you don't want to.

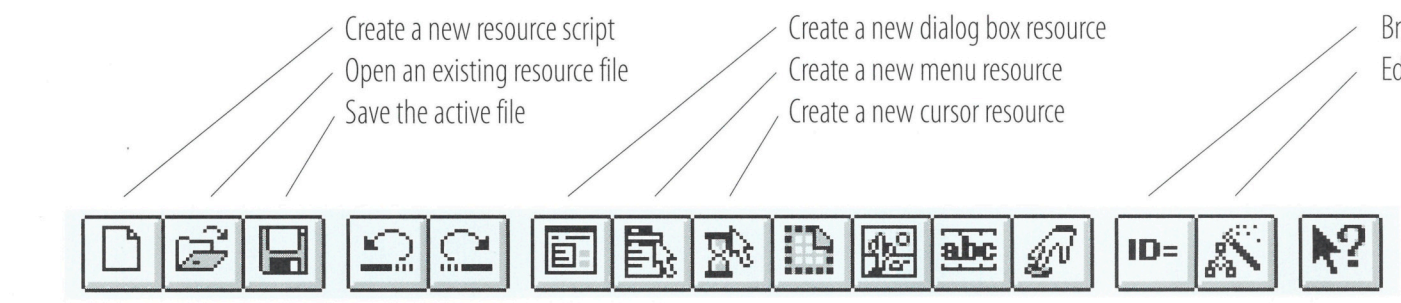
SCOTT RANDELL

MFC 2.0

Modern Art

DRAW YOUR USER INTERFACE

App Studio is a powerful resource editor that lets you draw common user-interface objects by directly manipulating the objects on screen — in other words, visual editing. App Studio also lets you incorporate Visual Basic™(VBX) controls (your own or third party's) into your interface. One of the new and powerful features of App Studio is its integration with ClassWizard, which attaches user-interface objects to code.



For more information on App Studio, see the *App Studio User's Guide*.

"Visual C++ does something that you've never been able to do before: it lets you draw your C++ application's user interface and connect these user-interface objects to code, and that's a real step forward. **We've tried to make these tools all use as much direct manipulation as possible,** so that when you're creating a menu bar, you *see* the menu bar, you *rearrange* the menu bar by clicking on a drop-down menu and dragging it to a new position. It used to be that you used RC script files, you'd write code, you'd compile it, you'd see your application interface, and you'd say, "Oh, that looks okay, I guess." Now with App Studio you have a powerful set of editors that let you make your interface look exactly the way you want it to look.

Another really exciting feature in App Studio is the ability to use VBX controls. These are reusable controls that can be used in both Visual Basic and Visual C++ programs. In the past, when designing your application interface, you'd have a certain set of standard Windows controls that were available to you. **With Visual C++, you can select from hundreds of VBX controls** that are commercially available — charts, grids, 3-D widgets, database controls, multimedia controls, and many others — and use them in your interface just like you'd use the standard Windows controls.

App Studio also has the ability to **visually connect your interface objects up to code.** It used to take time to learn all the Windows messages. You'd have to read documentation to figure out what messages you needed to handle and then take special care to interpret the message parameters correctly, but now all you need to do is select an object in App Studio and then open up the ClassWizard. It will show you all the messages that will be sent, so you can select the one that you want and it will create a member function for you automatically. It'll actually show you that message with a little hand graphic next to it, so you can visually see what messages are being handled and what functions are handling them."

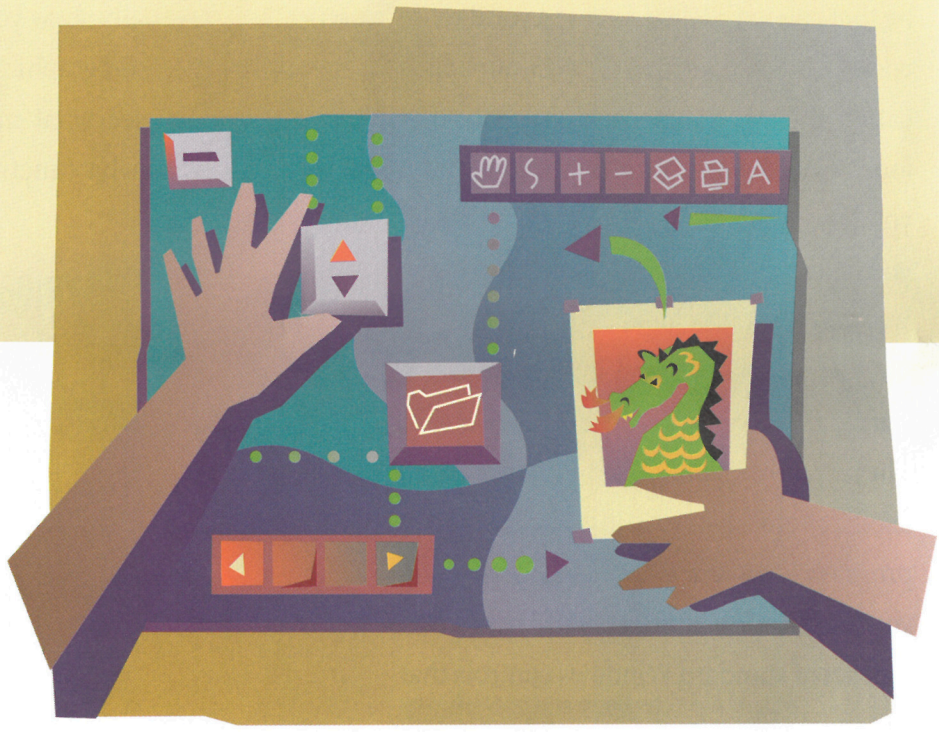
nbols in the active file
and connect resources to code

We're all Windows programmers in this group, and we got together to make it easier to create Windows apps. Something that seemed really compelling to a lot of us was the idea of being able to draw your interface instead of having to write code to do it — in other words, to do visual things visually. The notion of App Studio was born out of several people inside the group here who said, "Boy, Microsoft's got to do something like this; it's a much better way of creating applications." And that was the formation of the AFX group that created App Studio, about three years ago. We think using App Studio will make your work easier and faster, and it'll allow people who are less technical and more artistically inclined to participate more directly in the development process. We're excited about it.

CLIF SWIGGETT

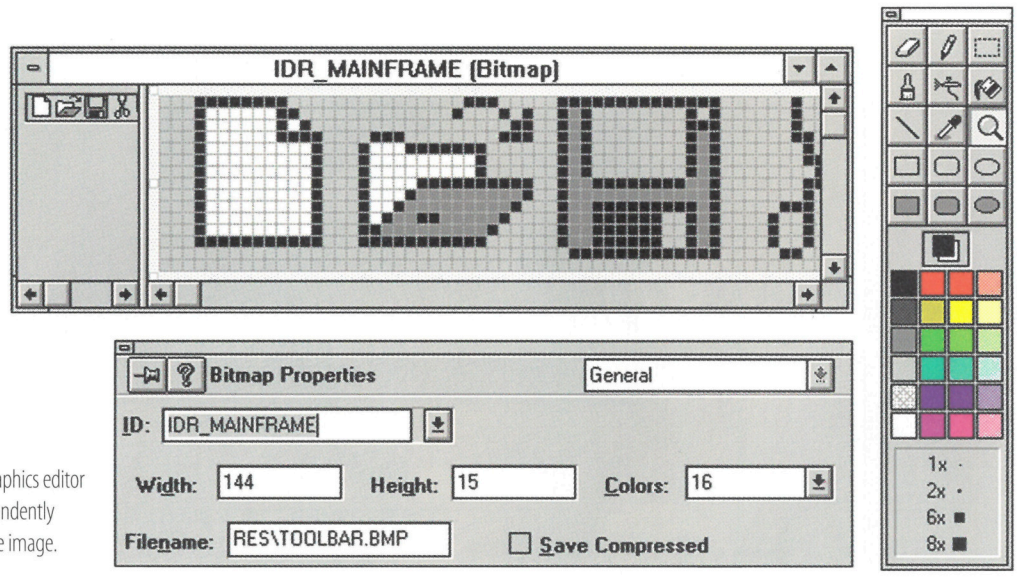
App Studio

Power Tools



DRAG-AND-DROP EDITING

Three important editors in App Studio are the dialog editor, where you can build and test a dialog box that very closely resembles the one in your finished application; the menu editor, which makes it easy and fast to build and revise even a complex menu structure; and the graphics editor, a full-featured editor geared to the professional developer.



The Visual C++ graphics editor displays two independently scalable views of the image.

For more information about resource editors, see Chapters 3, 4, and 7 of the *App Studio User's Guide*.

"When you're building a complex user interface you want your programming tools to be as easy to use as the interface itself. That's why we took a lot of time to make sure that wherever possible in App Studio **you work with your resources directly in the editor,** positioning with the mouse, dragging from one location to another, and so on. In fact, using drag and drop you can do tasks like building a dialog box, transferring controls between dialogs, arranging your menus exactly the way you want, even moving resources from one resource script file to another. You can even drag an icon or bitmap from a file manager window and drop it into your resource browser window, and bingo, the icon is added to your resource file. Try it!

Dialog Editor

One thing about our dialog editor that I find exceptional is that I have yet to see another dialog editor that is as full-featured as ours, yet uses the real controls that you're working on. It seems like there's often a trade-off between getting a full-featured dialog editor and getting one that uses real controls (not just bitmaps). It's a big advantage to always be able to see exactly what it's going to look like at run time.

Another feature that I really like about the dialog editor is the visual grid, which I don't think any of the other editors do. Also, having all the alignment tools and the test mode right up in the toolbar, right above where you're working, is a really efficient way to work. **When I'm laying out dialogs, I can just go drop controls and move them around and click around on the buttons** — I don't have to go searching through menus for the commands or look at some palette of unrecognizable icons way off in the distance.

Menu Editor

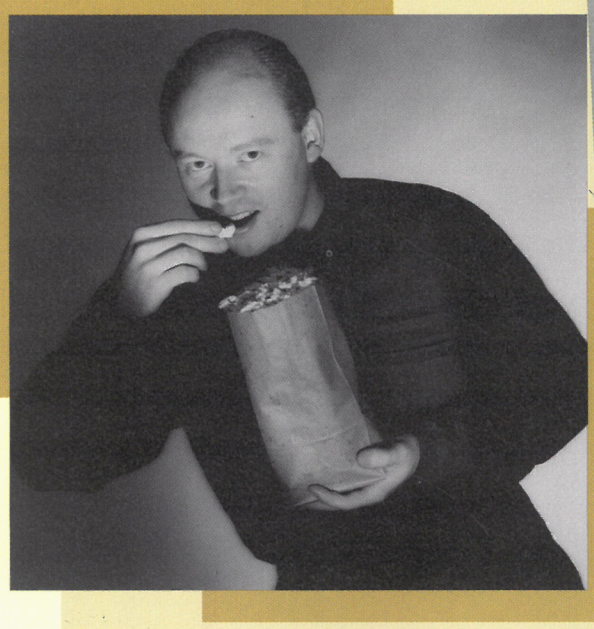
We designed the menu editor to make it very easy to rearrange your menu structure and to be able to select portions of your menu and just drag them around wherever you want them. As you move menu items around you find that what you're really doing is moving a set of related items. If they're connected to code, we don't break the connections to code, we just move them, we can rearrange the structure. **For example, you can turn a pop-up menu into a drop-down menu, or a submenu into a top-level menu or vice versa,** or take menu commands from one drop-down menu and move them to a different drop-down menu — just by dragging it and moving it around, and without changing any of the connections to code that you've made. And so it gives you a lot of flexibility to, at the last minute, decide that you don't like the menu structure and change it all.

Graphics Editor

I think our graphics editor is second to none — it's really geared toward the professional developer. Some of the things that it does that other editors don't do are things like having two views on the object at all times that are always kept in sync, being able to scale either one up or down, so you can, for instance, have one zoomed partially in and the other zoomed all the way in so you can look at things from two perspectives. **And it also does a lot more than the competition's graphics editor as far as brush styles, airbrushing, filling, being able to pick up colors** — picking up colors is especially nice, I think, because so often you'll want to go from one area of the image to another and instead of saying, "Oh, what color did I have? I can't remember," I can just go pick up the eyedropper and suck up the color and use it somewhere else.

One of the biggest problems when you're dealing with icons or cursors is — well, mainly with icons — is getting the screen and inverse colors correct. In our graphics editor, the inverse color is just another color on the palette, and you can click on it and draw with it. And I think that's something that an experienced programmer will really appreciate.

One of the neatest features we have is the ability to design a toolbar in the graphics editor using what we call using tile grids. This basically turns the graphics editor into a toolbar or palette editor, where you can edit little tiles of graphics that get put in a larger bitmap. This works very cleanly with the Microsoft Foundation classes to give you toolbars at the top of your application that you can customize with our image-editing tool. And App Studio makes it very easy to move toolbar buttons around, to change the width of them and have them instantly reflected in your app."

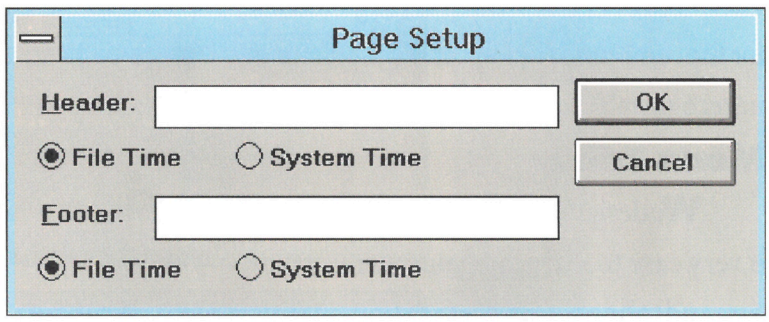


GARTH HITCHENS

Usability

WHO CARES? YOUR CUSTOMERS

With all of the powerful tools Visual C++ provides for creating Windows-based applications, you still face the task of designing an application that is right for your user. The design phase can be challenging. How do you use color in your application for best effect? How should controls be arranged on a dialog box for ease of use? Fortunately, designers have answers to these and other questions about interface design.



This simple dialog shows good UI design. The text is concise, yet descriptive. The Header and Footer groups share a common layout that reinforces their similar function. The buttons' size, shape, and location reflect their related purpose. And the overall layout is compact, but not cramped or cluttered.

What are your concerns about using color in interface design?

Color is a very complicated phenomenon in terms of how humans perceive and get information from it. People have very personal reactions to color — it can really make us feel specific things, not rational things. For example, when you see a lot of bright colors together, your reaction generally is that this is something for children — you get a sense that it's sort of cheap. If you think about advertising material, the stuff on gray paper with only one color or two — maybe black, seems more expensive and of higher quality. Because people like color a lot when they're designing Windows applications, they tend to use a lot of color — perhaps without realizing that a lot of color makes people think that the thing is cheap or poorly put together. People will have strong feelings about all colors, except for blue and probably red. That's why you should use color as little as possible in an interface, and wherever you do use color, you must always let the user change it. You should never hard-code a color because there will always be somebody who hates that color.

How can programmers learn about good interface design?

One book is *The Windows Interface: An Application Design Guide*, available from Microsoft Press. This is the best reference that you can get about good interface design for Windows. It tells you everything you need to know. If you get the book, you can also get a disk which has some online pieces on it, including an interactive guide that shows you the rules in the *Interface Guide* in a running app.

Do you have any practical advice you could give about how you, as a designer, deal with multiple display devices?

Microsoft has done some surveys to find out the percentage of various display devices in use, and we found that very few people who run Windows—less than 10%—have CGA or EGA displays. Generally, when people upgraded to get enough memory and a faster chip, they upgraded to VGA. And a lot of the LCD displays—in fact, all the current LCD displays in portables—use a VGA display writer. So what we do is design for VGA, and then we let that same design run pretty much as is on higher-res machines. In a high-res machine, like Super VGA, the pixel is still a square pixel and it's smaller than the VGA one. Everything will run, but everything will be smaller. And it seems to work pretty well. The fonts are proportional, and so everything goes down in size proportionally.

Any final words of advice?

Graphic design is a well-established profession. Graphic designers have a well-honed procedure that solves problems by producing a large number of initial ideas, reducing those to the best ones, further developing all of those—maybe three or four ideas, then reducing those some more, and eventually coming up with a good design. When designing an interface, programmers should think about hiring a graphic designer. If they're a small shop, they don't have to spend a lot of money on an interface designer. If all they do is pull in somebody visual—the person who's designing the documentation, the person who's designing the packaging, their marketing person, their advertising person—whoever it is, just somebody who has visual training, they will be ahead if they take the designer's feedback seriously. I think the main thing is to realize that graphic interfaces need to look the way they work and work the way they look. Don't get carried away by something cool, try to make something usable that people will understand.

VIRGINIA HOWLETT

Interface Design

Objects

A SILVER BULLET?

Object-oriented programming is more than learning a new syntax. It means a new approach to design, as well as implementation. By enabling greater code reuse, C++ can help you start designing today for the increasingly complex applications of tomorrow.

"We see virtually all C development moving to C++ over the next two to three years."

—Bill Gates

For more information about C++ and object-oriented programming, see the *C++ Tutorial* and the *C++ Language Reference*.

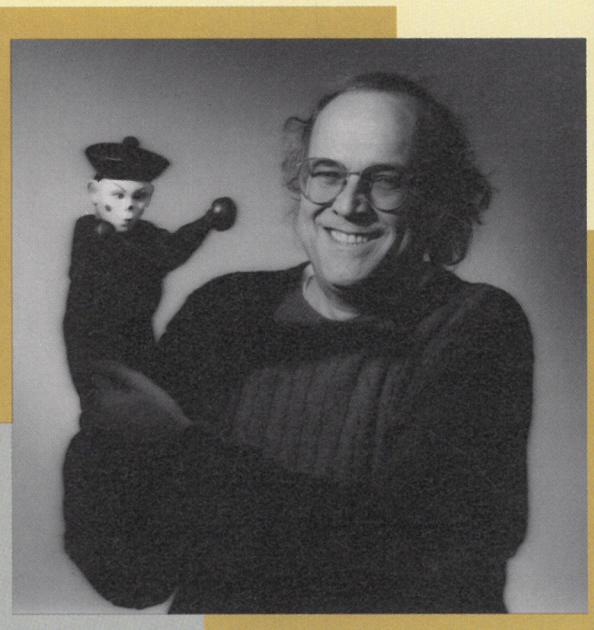
"The C++ language comes from AT&T, basically, the same spring from which came C. The primary difference is that C++ extends the notion of data structures, so you can associate functions with data. That kind of type binding gives you an encapsulation of functionality. And it has overloading of operators and functions, which allows you to express algorithms in a way that's relatively type-independent. This sort of technology is called object-oriented programming, which relates to data abstraction and encapsulation.

It's possible to do this in C — in fact, a lot of C++ translators generate C code — but there the binding's merely a convention. For example, I can write you a C header file where a bunch of functions act on a bunch of structures, and the convention is that you use those functions when you want to operate on the data. But there's nothing to stop you going in and partying in true C-programmer style on any part of it. So, object orientation is possible in C, but it's a personal discipline, whereas in C++ there's a formal binding between functions and data, and it's a natural way of expressing yourself. C++ makes it easier, I guess, to do things right.

This object-oriented approach is not a silver bullet — we all know that's a fiction — **but it does give you leverage by making your code reusable in new contexts,** and that's where I see the greatest productivity gains. It can reduce the difficulty of programming complex applications. If you look at applications, I think that they're moving toward a component architecture. For example, consider Excel. It grew out of a relatively simple idea called a spreadsheet, which was initially implemented in assembler on a 16-bit machine. Today, Excel is a gigantic piece of code that has all *kinds* of things in it — charting, forms packages — you name it. Applications are so rich and so complex that they're becoming prohibitively difficult to write. And so the approach being adopted is to separate the functionality. Instead of every application having its own charting package, they all use common services. You use interfaces like OLE, or a library interface, to partition your product into components. **Object-oriented programming is ideal for a component-based architecture,** because the encapsulation exposes only the parts you really need to know about. And it gives you the flexibility to change the parts you don't know about without affecting the body of the code.

I think the development community is pretty much committed to C++ and to object orientation. As for learning C++, well, the language itself is conceptually simple. But you have to understand that object orientation is a new approach, not just different syntax. If you approach it from a C programmer's perspective, you'll take a long time to grasp what's going on. **The experienced C++ programmers, as far as I can observe, think in terms of classes,** not in terms of algorithms or functions. You design the program in terms of the classes it needs to operate. So to learn C++, you must understand the benefits and the need, as much as the nuts and bolts which you'll find out quickly enough in the process of putting together a real application. Yes, there's complexity in the nuts and bolts, too, but it's that basic understanding of what an object-oriented program looks like and how it's constructed — that's what you must have, and the rest will follow."

DAVID JONES



C++ doesn't cure all ills. It's more like a power tool. If I give you a power planer, you'll probably use it, because the benefits over hand planing are obvious. But you can still make mistakes. When you add a powerful programming tool, it's still a human idea being expressed. Programming is the expression of ideas in logically tangible form, and because it's done by human beings, and because it's a creative process, it's flawed. You never get it right the first time. No tool can eliminate that entirely, but you still naturally choose the better tool.

C++

Throughput

CHOICES, CHOICES, CHOICES

Compiler technology is the engine under the hood. Visual C++ offers a mature, comprehensive suite of optimizations that let you tune your code for maximum efficiency. Compiler technology also includes throughput features like precompiled headers, which can dramatically reduce compilation time.

For more information about improving program performance, see Chapters 1-5 of *Programming Techniques*.

"The compiler technology raises a lot of interesting questions: "How do I get my app to compile fast? How do I get it to run fast? How do I minimize size, or get debug or browse information?" The Visual Workbench handles a lot of that stuff, but it's nice to be aware of *all* the things the compiler can do for you.

To compile quickly, precompiled headers are first and foremost. Now we give a more convenient form with the /YX option, where we automatically create the precompiled header. Just throw this switch, and we'll precompile whatever we can. You don't have to think about it. For more control, use the /Yc and /Yu combination. The best way to use these switches is to create a header file — it can even be a .C file — that contains only the headers you want precompiled. This fits nicely into a makefile, and it can be very fast, although it means a little extra work.

While you're in the middle of development, **use our fast compiler.** It's *much* faster in turnaround time, and you can use precompiled headers to speed up even more. When you are ready to test the release builds, move over to the full optimizing compiler and try increased optimization.

Optimization, you know, is always an experimental thing. What works best for my app? It varies a lot, but some things are almost always worth trying. Global enregistering, with /Oe , can both save space and increase speed, because it tries to keep more things in registers. And it's always helpful if you can assume a 286 as your base machine. That's the /G2 switch, which gives much better speed and some reduced size. For Visual C++ , we added /G3 , which lets us selectively use 386 instructions for better code generation, if you're willing to assume a 386 machine. This can reduce size as well as increase speed. And if you can avoid aliasing in your code, the aliasing options — /Ow for Windows and /Oa for DOS — tell the compiler to make optimizations it couldn't do otherwise, because of the nature of C and C++.

It's usually worthwhile to profile your app and say, "Where am I spending my time? What things are really critical to make fast?" Once you have them algorithmically as fast as possible, then throw speed optimization onto that, either by using the command-line switches or pragmas in the source files.

In places that aren't speed-critical, you might just as well compile for space and reduce the size. Size is critical in Windows apps because it's a swapping environment, so reducing your app's size might actually speed it up, because it swaps less. And that's where p-code may come in. P-code lets you take parts of your app — particularly the user interface — which are generally not speed-critical, and reduce the size dramatically. So your working set that stays in memory is much larger, you have less swapping, and the overhead of executing in p-code isn't noticeable because you're mostly waiting for human interaction. But again, p-code is a selective thing, and you'll want to do some profiling first.

There are so many other things. Inlining — we have multiple levels of that. We have intrinsics for common runtime calls, we have loop optimizations, we have pragmas that let you tune on a per-function level, if you wish. Sure, this might be more work, and it's not needed for many apps, but if you need the ultimate control, we provide it."

We've been at this a long time, you know. Our compiler technology is mature, our optimizations are far broader than the competition's, and there's been a dramatic increase in the robustness and stability of the optimizer over the last few releases. Yet we still generate the best code in the industry across a broad spectrum. I mean, there'll always be cases where somebody else can handle a particular application better. But over a wide range of applications, we still have the best.

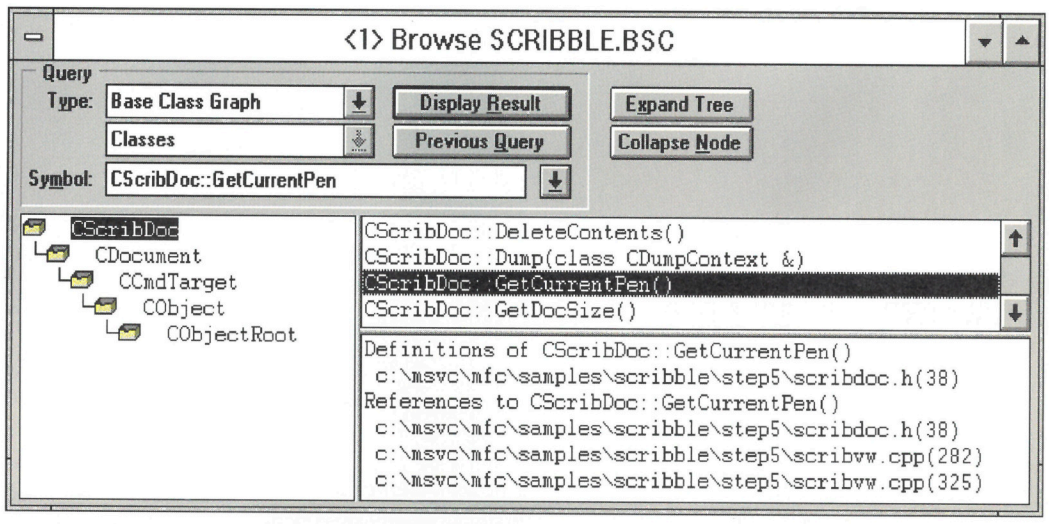
DAVE WEIL

Relationships

THE PRIVATE LIVES OF CLASSES

The Visual Workbench browser makes it easy to navigate your code.

From within a source file, with a simple menu click or shortcut keystroke, you can jump directly to a symbol's definition or visit each place in every source file it is referenced. And in the Browse window, you can query and display class hierarchy graphs and function call graphs, as well as definitions and references.



The Visual C++ browser helps you understand complex code relationships and work efficiently.

For more information on the browser, see Chapter 10 of the *Visual Workbench User's Guide.*

"You want to know what the browser can do for you? Okay, I'll give you an example of what I was doing yesterday. Here I am looking at this function, and I don't quite understand it — I wasn't sure that this structure was being used correctly. So I just put my cursor on this global variable, and I hit F11, and there was the definition of that variable. So now I know the type. Then I moved the cursor over to the type and hit F11 again and boom! There I was at the structure definition, so I can look at the members and so forth.

As it turned out, there was a problem with the usage. Having looked at the members, I saw that it wasn't being done quite right (there was a comment there that helped me), so I used the Pop Context feature, and there I was back at my initial global variable reference, and I made the fix there to use the field correctly. Then I hit Shift+F11 on the global variable and I visited all of the references to that variable to make sure that the same construction didn't appear anywhere else.

So it was all pretty simple — I figured out how the variable was supposed to be used and then fixed it. And I have no idea which files I visited. I suppose I could go and look at the logs to see where I checked in changes, but other than that, I don't know — I just fixed it. Which is the way it ought to be.

You can also use the browser to find a particular member name— you just type it in the Name field and it shows you all the possible overloads of that name. Or maybe you just want to see all the places where that name was defined, which is an end unto itself, right? You can see all the places where it's defined just by typing the member name in that box.

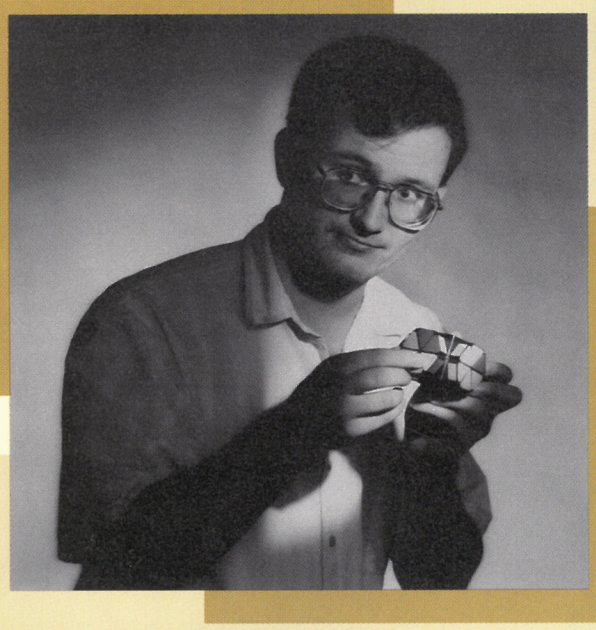
You can also use the browser to query graphs. There are four kinds of graphs available — class graphs, call graphs, function graphs, and caller's graphs — but of these, the one I use the most is probably the one that seems most esoteric — the caller's graph. Let me tell you how I used that yesterday to fix a bug.

Once again, I was in code that I'm not terribly familiar with. There was a problem with the searching code, and I had no idea where to even start. I knew I needed to make a change to one of the window procs, but I had no idea what the name of it was or what file it might be in. But I knew it had to do with searching, and I figured, well, ultimately, there's some code there that's calling some find function.

So I opened the Browser window and I picked the reverse call graph, and I said, "Show me all the functions called FIND*," figuring that there was bound to be some worker function down there called FIND-something, and I got a dialog box that let me pick the one I wanted. So I picked the one that looked like it was the most primitive — it was called FINDNEXT or something like that — and that became the base of the caller's graph. And then I said, "Do the query," and I expanded it fully.

So now I had all the places in the program by which you could reach that primitive worker function. And I *knew* that it had to do with finding, so I knew, ultimately, we were on the right track. And then I looked through that call graph until I found the thing that looked like a window proc. I double-clicked on it, and presto! I was in the function that I needed. So my input to this whole thing was just guessing, "Well, the function's probably called FIND-something," and the rest was just following the dots.

One last point — **Visual C++ is the first Microsoft product that lets you browse while debugging.** This is really handy because it lets you begin a browser-assisted autopsy from the site of a crash. If you run your application and get a Windows RIP, the debugger catches this and you get placed at the offending line. If you don't understand the offending line, you can quickwatch the variable to see its value and use the Go To Definition/Reference Command to see how it's defined or used. The debugger and browser have good synergy."



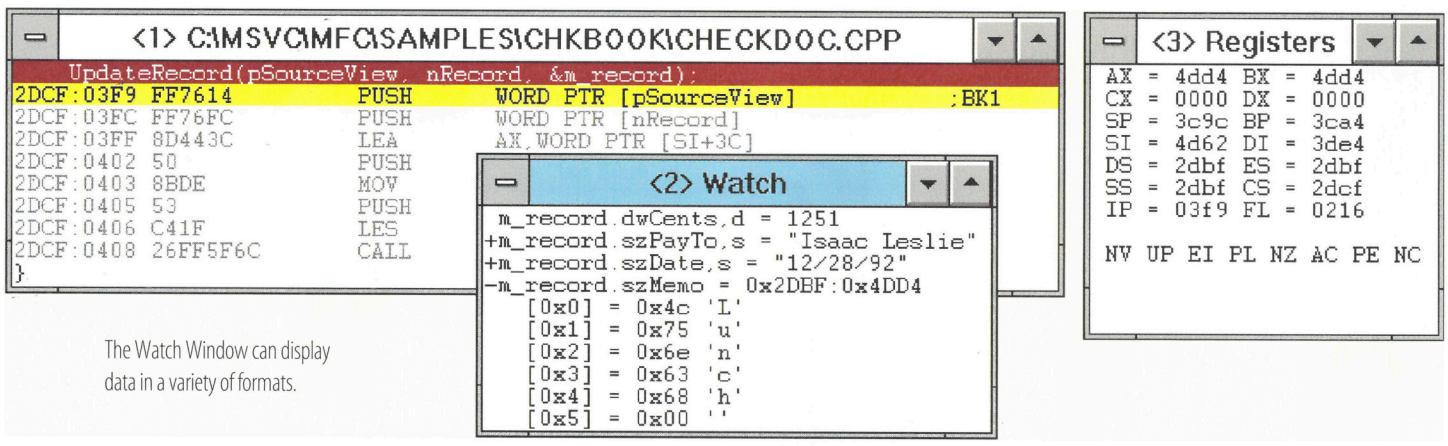
RICO MARIANI

Browser

Problems

WHEN BAD THINGS HAPPEN TO GOOD CODE

Visual C++ gives you different debuggers for different needs. The integrated debugger runs inside the Visual Workbench, offering great convenience. CodeView, available in Windows and MS-DOS versions, runs separately and offers extended capabilities. Together, the Visual C++ debuggers make a very powerful combination..



The Watch Window can display data in a variety of formats.

For more information on debugging, see Chapter 11 of the *Visual Workbench User's Guide*, and the *CodeView Debugger User's Guide* (Visual C++ Pro users only).

Joel: The integrated debugger probably gives 80% of what you need for everyday Windows debugging. And it's just a more natural way to work. You don't have to exit Windows or flip screens. **The debugger is right here, running in the same environment as your app.** And this provides great convenience, as well as flexibility.

Say you're in a source window and you want to check the value of a new variable. You just put your cursor on the variable name, or double-click to select it, then press Shift+F9. The QuickWatch window displays the new variable, and if it's a structure, you can expand it to examine the members, and so on. We know different people have different working styles, so we actually provide three different ways to get to QuickWatch — the way I just described, plus the menus or the toolbar. And the same is true of other features.

Running under Windows gives you a lot of versatility, too. **You can start and stop other Windows apps** while the debugger's running. For instance, you want to launch the Spy application to get more information. Go ahead, it's not too late. Or, you can change things in the debugger while your app is running. Let's say you want to go back and set a new breakpoint. Again, it's not too late. You can set the new breakpoint immediately, without waiting for your app to terminate or reach a breakpoint."

Mark: Yeah. Of course, some cases really demand extended capabilities, like debugging a DLL or whatever, and that's where CodeView comes in. We have two CodeView versions, for Windows and MS-DOS, and they have an identical interface, so you don't need to learn two sets of commands.

In CodeView, you really own the process, and you can reach down to manipulate things at a low level. You can enter new data directly in memory. You have extra commands, like Dump Modules, specifically for Windows programming. And the command window lets you do all kinds of things once you stop at a breakpoint. You can dump a variable or some registers — do benign, information-gathering things — or you can change things, if you really know what you're doing. Change the instruction pointer to jump around in your code. Or type in assembly instructions, like MOV AX, 0, and actually rewrite your program at run time. This is not for the faint of heart, like I said, but if you have the need and you know what you're up to, CodeView provides the means."

Joel: Right. It's not a question of which debugger is better, but what you need at a given time. Integrated debugging is fast and it blends very nicely with the other Visual Workbench capabilities. But other times you must have those extended features, so you fire up CodeView because that solves your problem in the most efficient way."

MARK BRODSKY
JOEL MERLIN

Product Support

WHO YA GONNA CALL?

Customer support is more than just phone calls. Beyond the traditional phone service, we're expanding electronic support in a variety of ways. We're improving the depth, as well as breadth, of our services. And we're constantly asking customers what we can do to serve you better.

The overriding theme here is quality versus quantity. We're striving not only to provide technically correct and accurate answers, but in providing the answers, inject a bit of personality and convey strong customer-service skills, good communication skills, good presentation ... so that it's excellent, as opposed to good.

LORI REYNOLDS

For more information on product support services, see the Introduction to the *Visual Workbench User's Guide.*

"One of the big areas that we'll focus on is electronic support. Phones work well to get people up and running, to get them set up, installed, configured. But we find that electronic support is the better way to support customers in the long run.

Right now, electronic support primarily means CompuServe. We've expanded the MSLANG forum, adding more information and a better breakdown of sections, and we put our best engineers up there supporting customers. CompuServe also lets customers access other resources such as the Knowledge Base, the Software Library, information from the MSDN group, and other forums — both the Microsoft forums and the ones from other companies. **Electronic support lets us route the question to the best person to answer it, not just the first available engineer in a phone queue. And that makes a big difference.** Finally, with electronic support, sample code can be included in questions and answers, and that's something that you just can't do over the phone. When you get down to it, that's what represents the problem and the solution, in most cases.

In the future, we're considering expanding to other bulletin boards after we feel that we know how to do it right. We really feel it's the best way to support our customers. We've found through surveys that over 80% of our customers have modems. One of our goals is to figure out ways to assist and provide direction to that other 20% so that they can access that electronic support.

Right now, we need to do a better job of communicating to customers what services are available and how they can access them. They've got the phone, they've got the CompuServe MSLANG forum and other forums, MSDN, they've got the MSDL, which is the Microsoft Download board where a lot of information is available seven days a week, 24 hours a day. The Developer Service team is a group specifically for the non-technical needs of developers. **So there are a lot of resources — we just need to clearly show customers where they can go and how to get there.**

One of the things that we can do much better with Visual C++ is to help customers get up and running faster. Because we *always* deal with a tremendous number of brand-new issues after a product ships, we're implementing training specific to setup, and it'll be at the tail end of the product's beta cycle, so that we're fresh with that knowledge when it ships. We're going to train and prepare a setup SWAT team to handle the new things that come up. Related to that, also, is the new setup program... we need to make the actual installation easier.

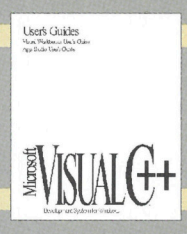
In general, we need to listen better. I think we're doing a better job, and we'll continue to improve on it. We need to continue to do surveys, report customer feedback back to the product group, and just listen and incorporate. We've made progress, and we'll continue to refine and redefine our support offerings. What customers need today is quite different from what they needed two years ago, and we need to react to those changes and meet their needs."

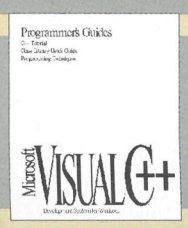
Customer Support

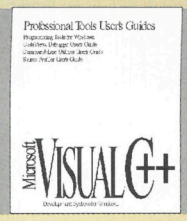
A COMPREHENSIVE

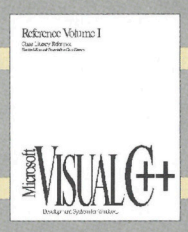
A quick guide to the Visual C++ print documentation.

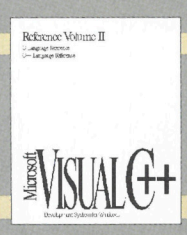
What PRINT COMPONENTS

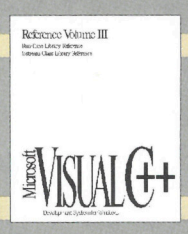


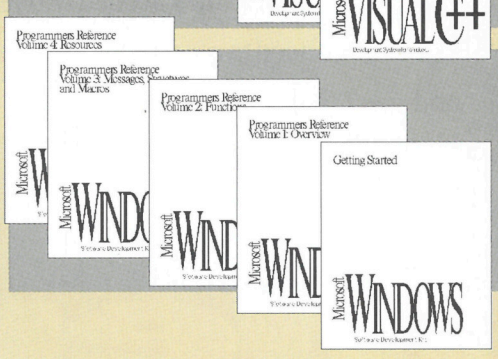












Documentation	Standard	Pro	C7 Update
Introduction			
Presenting Visual C++	■	■	■
User's Guides			
Visual Workbench User's Guide	■	■	■
App Studio User's Guide	■	■	■
Programmer's Guides			
C++ Tutorial	■	■	
Class Library User's Guide	■	■	■
Programming Techniques	■	■	
Professional Tools User's Guides			
Programming Tools for Windows		■	
CodeView Debugger User's Guide		■	
Command-line Utilities User's Guide		■	
Source Profiler User's Guide		■	
Reference, Volume 1			
Class Library Reference	■	■	■
Reference, Volume 2			
C Language Reference		■	
C++ Langauge Reference		■	
Reference, Volume 3			
Run-Time Library Reference		■	
iostream Class Library Reference		■	
Other			
Comprehensive Index		■	
MFC Quick Reference Card	■	■	■
C/C++ Version 7.0 Update			■
Windows 3.1 SDK			
Getting Started		■	
Programmer's Reference, Volume 1: Overview		■	
Programmer's Reference, Volume 2: Functions		■	
Programmer's Reference, Volume 3: Messages, Structures, and Macros		■	
Programmer's Reference, Volume 4: Resources		■	

Note: Additional Windows SDK books are available. See your SDK fulfillment card.

TOPICAL GUIDE TO DOCUMENTATION Where

General Information

If you want to:	*See:*
Edit and debug programs	*Visual Workbench User's Guide*
Use command-line utilities	*Command-line Utilities User's Guide and Tools Technote Viewer*
Locate specific information	*Comprehensive Index*
Look up an error message	ERRORS.HLP
Get answers to commonly asked questions	Choose Obtaining Technical Support from Help menu
Configure and optimize your system	*Visual Workbench User's Guide*
Learn about using Help	*Visual Workbench User's Guide*

C Programming

If you want to:	*See:*
Understand C syntax and usage	*C Language Reference*
Use the C run-time library	*Run-Time Library Reference*
Optimize programs	*Programming Techniques*
Manage memory	*Programming Techniques*
Compare Microsoft C with ANSI C	*C Language Reference*
Write portable programs	*Programming Techniques*

C++ Programming

If you want to:	*See:*
Learn C++	*C++ Tutorial*
Use Microsoft Class Library	*Class Library User's Guide, Class Library Reference*
Program with C++ and Windows	*Class Library User's Guide*
Understand C++ syntax and usage	*C++ Language Reference*
Compile code written for other C++ compilers	*C++ Language Reference*
Manage memory	*Programming Techniques*

Windows Programming

If you want to:	*See:*
Program with Windows and C++	*Class Library User's Guide or Visual Workbench User's Guide*
Create a Windows-based application with AppWizard and ClassWizard	*Class Library User's Guide, Visual Workbench User's Guide, or App Studio User's Guide*
Use resource tools	*App Studio User's Guide*
Find reference information on the Windows API	*Windows 3.1 SDK Programmer's References, Volumes 1-4*
Get answers to commonly asked questions on programming in Windows	Open TIPS.HLP in the HELP subdirectory
Get Help on Windows tools	Help files in the HELP subdirectory

Special Topics

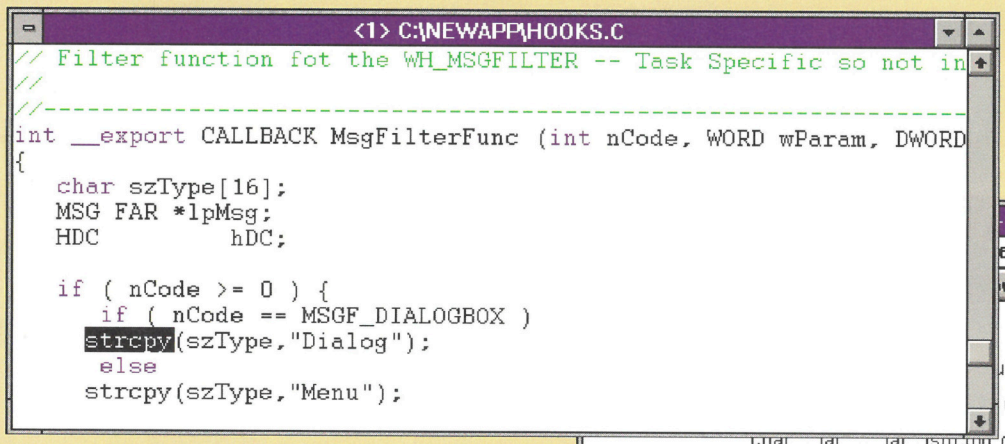
If you want to:	*See:*
Use p-code	*Programming Techniques*
Use precompiled headers	*Programming Techniques*
Profile your program	*Source Profiler User's Guide*
Use the graphics libraries	*Programming Techniques*

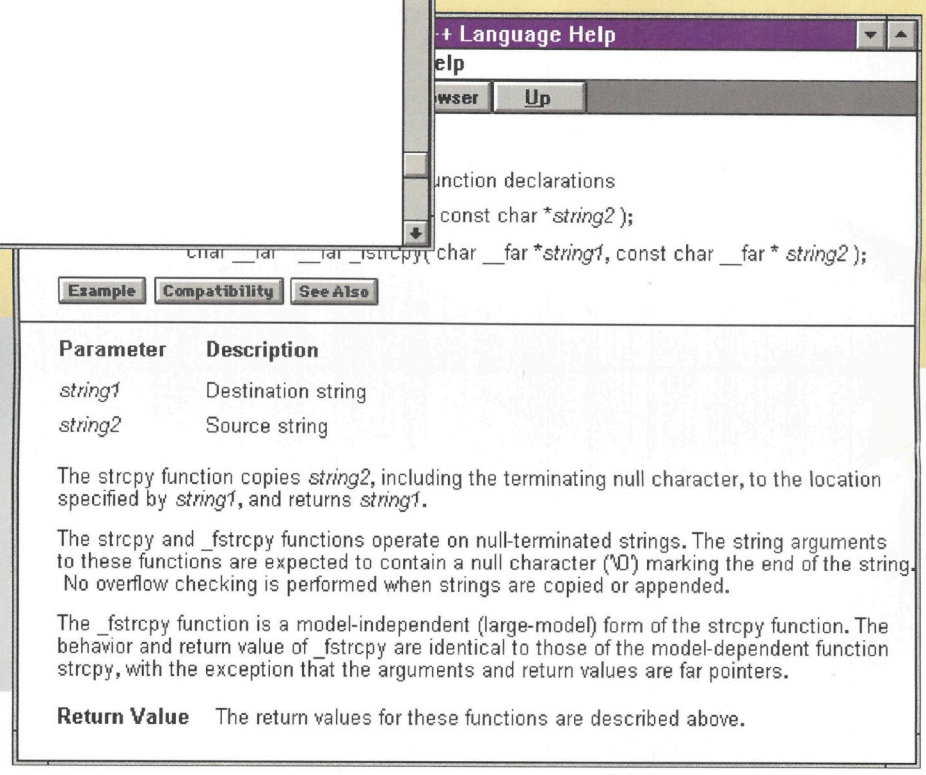
Print

Online Help

INSTANT ACCESS

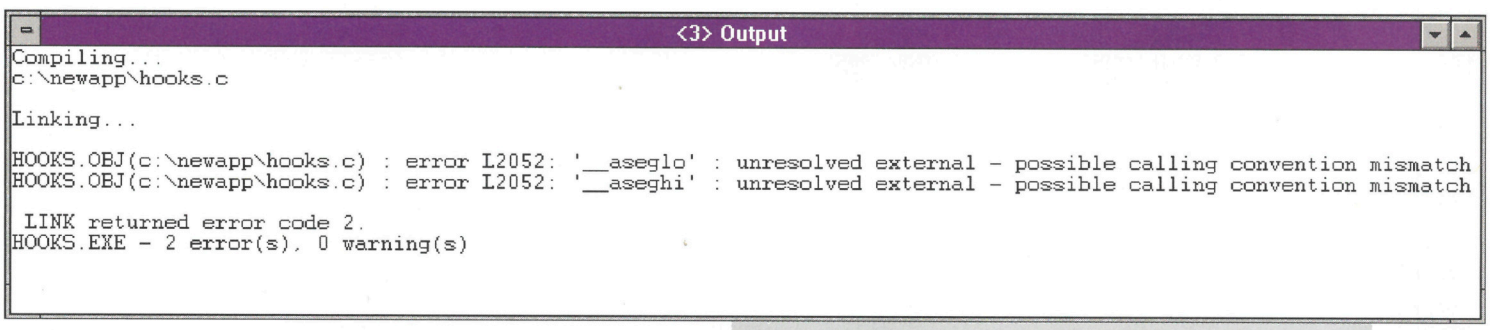
Use these techniques to get help anywhere in the Visual C++ environment.

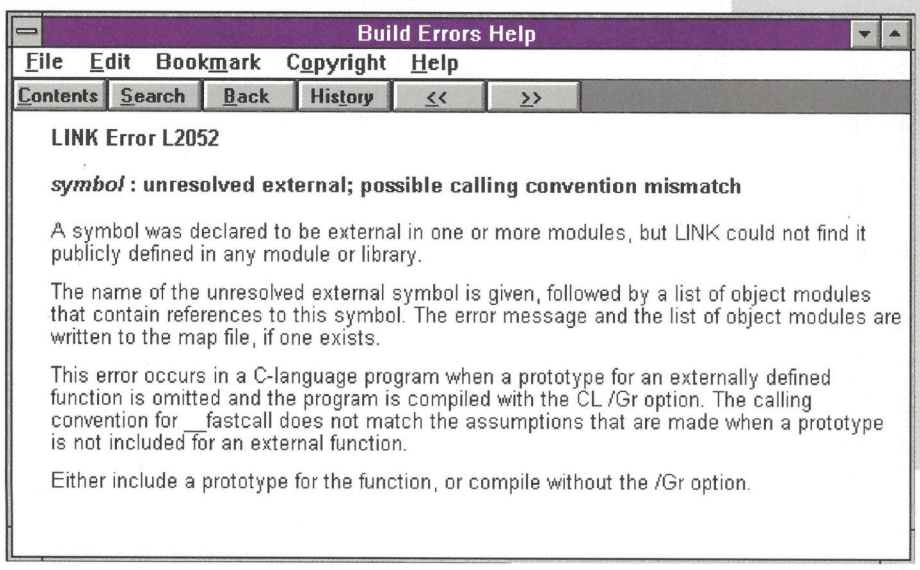




ON A FUNCTION NAME...

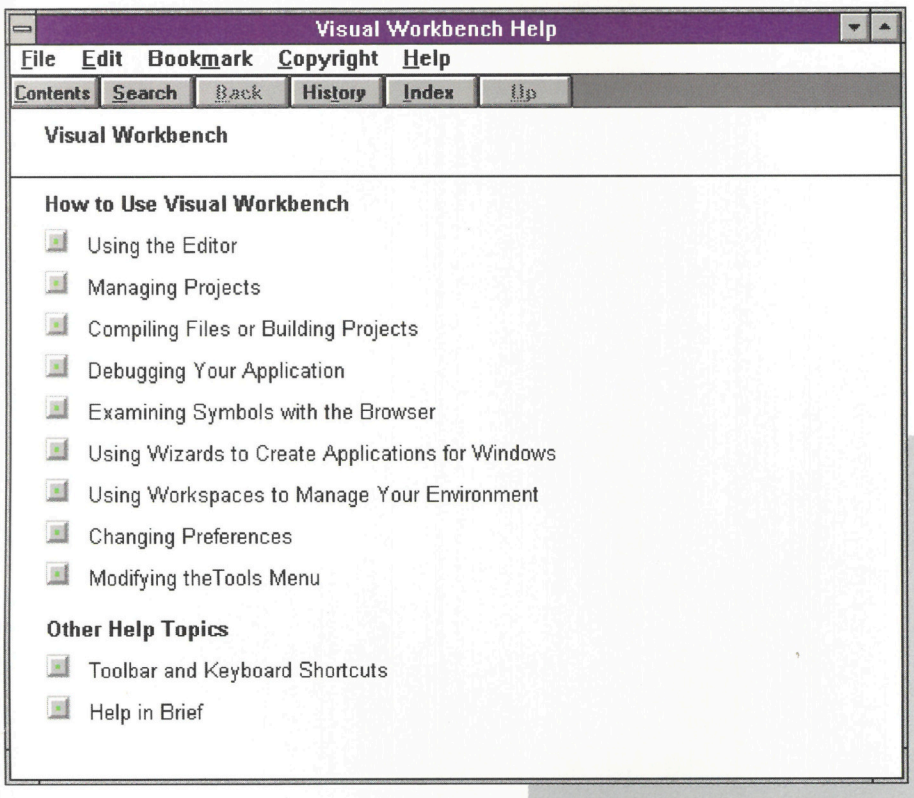
Press F1 with the insertion point on a function name or keyword in a source file.

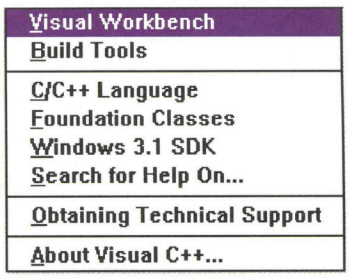




Press F1 with the insertion point on an error message in the Output window for information about that error.

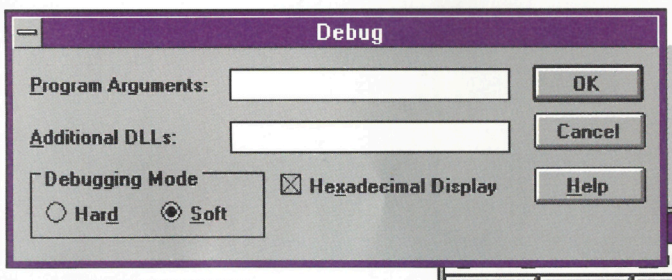
ON ERROR MESSAGES...





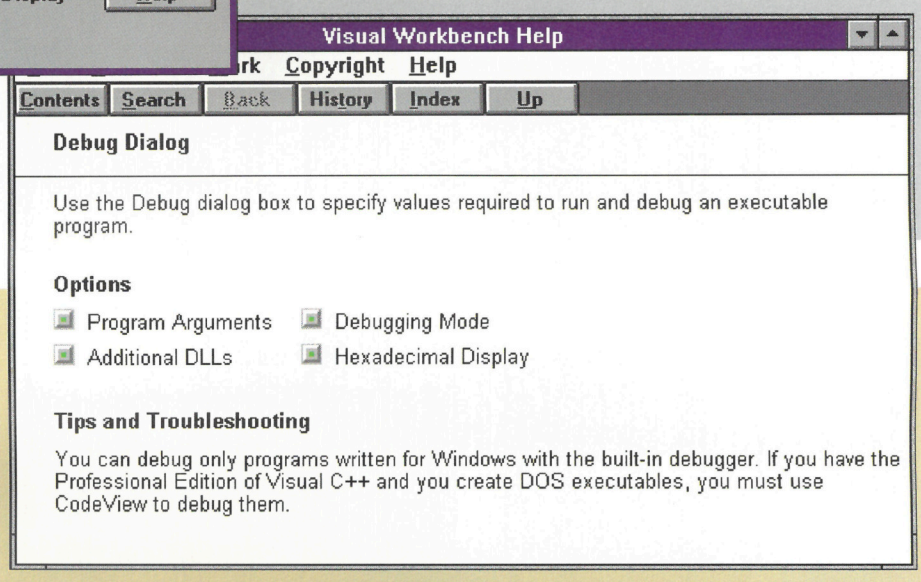
GENERAL HELP...

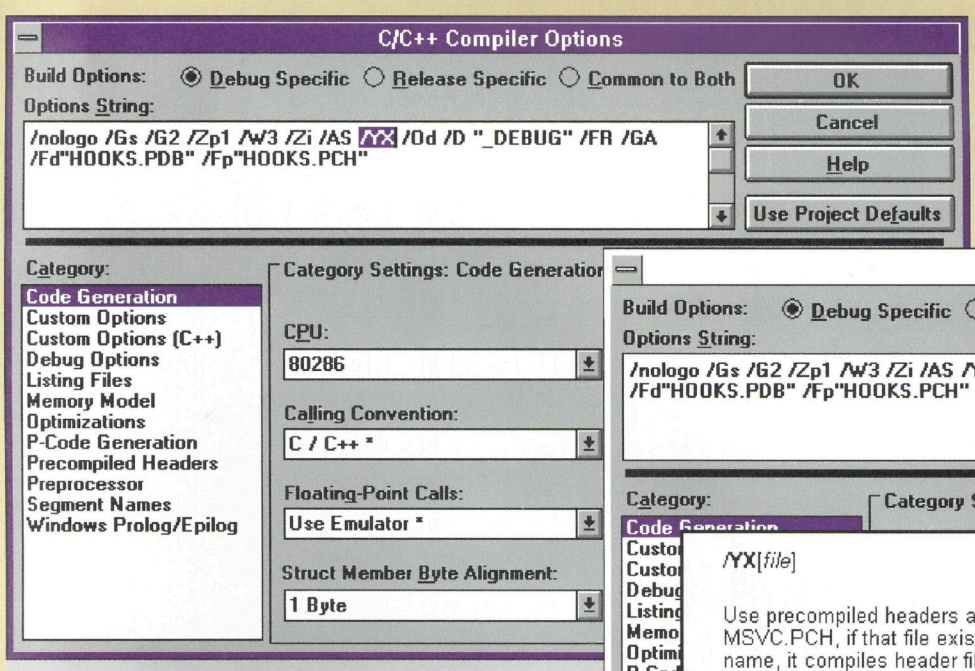
Use the Help menu to find general categories of help.



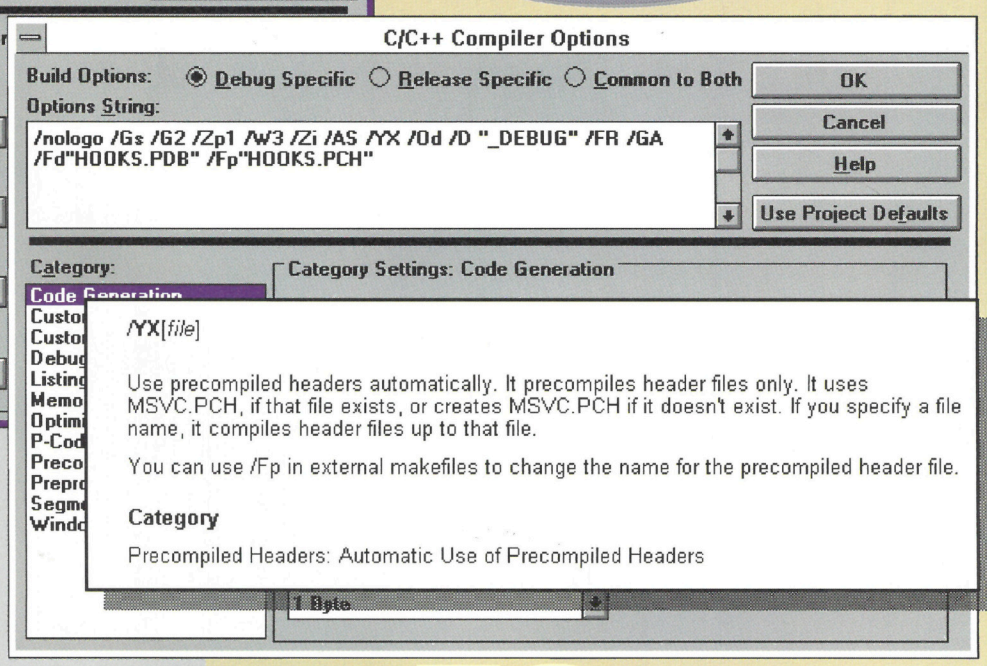
IN DIALOG BOXES...

Click the Help button in dialog boxes for help on options and for tips.





IN OPTIONS DIALOG BOXES...



In the Compiler and Linker Options dialog boxes, you can:

1. **Select an option in the Options String and press F1.**
2. **Select a control and press F1.**
3. **Select a category and press F1.**

Online

Teamwork

TRUE GRIT

Winning products don't happen by accident. Thanks to the entire Visual C++ team for their passion, creativity, focus, and plain old stamina.

DEVELOPMENT

PROGRAM & PRODUCT MANAGEMENT

PRODUCT SUPPORT

Printed in the United States of America.

Document No. LN29680-0193

"Presenting Visual C++" Creative Team:

Peter Lancaster: Lead / Guiding Light
Moira Macdonald: Editor-in-Chief
Phil Nelson: Writer
Tim Celeski: Designer/ Art Director
Leslie Newman: Illustrator
Keith Brofsky: Photographer
Interviews conducted by Phil Nelson, Peter Lancaster, Don Gilbert, Patrick Kelley, Chuck Sphar, Steve Ross, and Barbara Ellsworth.

If you want to share any comments about this piece, or about any other part of the Visual C++ documentation, write to Phil Nelson, User Education Manager, C/C++ Language Products, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399, or email: Internet: y-vc1doc@microsoft.com or CompuServe: INTERNET:y-vc1doc@microsoft.com